Schema Versioning in Conventional and Emerging Databases

Zouhaier Brahmia
University of Sfax, Tunisia

Fabio Grandi
University of Bologna, Italy

Barbara Oliboni
University of Verona, Italy

Rafik Bouaziz
University of Sfax, Tunisia

INTRODUCTION

Persistent information and data-centric systems use databases to store data. The contents of a database must adhere to a formal structure that is fixed in advance, and is called the schema of the database (Date, 2003). In those systems, not only data changes are obvious tasks done almost every day but also schema changes are unavoidable, in order to reflect a change in the real world or in the user's requirements, to correct mistakes in the initial design, to migrate to a new platform or to allow the expansion of the application scope over time. Two main problems have to be considered when dealing with any schema change: *semantics of change* (i.e., the effects of the change on the schema itself) and *change propagation* (i.e., the effects of the change on the underlying data). Resolving the former guarantees schema consistency, while resolving the latter guarantees consistency of data with respect to the changed schema.

In the literature, s*chema evolution* and *schema versioning* (Roddick, 1995; Jensen et al., 1998) are the two techniques that were proposed to support schema changes in a DBMS, without loss of extant data and with continued support of legacy applications. After applying schema changes, schema evolution keeps only the current schema version and retains the data which are adapted to such a schema. On the other hand, each time schema changes are applied, schema versioning creates a new schema version, while preserving old schema versions and their corresponding data. With schema versioning, data access through any schema version is supported, which avoids applications developed with past schemata to become obsolete.

Schema versioning has been widely investigated, both in the context of traditional and temporal database research. Several models, languages and approaches, dealing with schema versioning, have been proposed during the two last decades in the relational, object-oriented, and XML databases. However, to the best of our knowledge, limited support of schema changes and no support of schema versioning is provided by commercial database management systems (DBMS). Therefore, diligent database designers and administrators have to work hard to solve the problem of evolving a database schema in an ad hoc manner. Besides, with the growing use of emerging databases (e.g., multimedia, temporal, biological, and NoSQL databases), research work has also recently done on the problems of schema versioning in such settings.

DOI: 10.4018/978-1-5225-2255-3.ch178

Figure 1.

S1

AUTHOR(ID, NAME, PHONE, COUNTRY)

ID	NAME	PHONE	COUNTRY
1	Aicha	11223344	Tunisia
2	Cristiana	55667788	Italy

The main goal of this chapter is (i) to present the recent research proposals, not already covered in (Brahmia et al., 2015), that deal with schema versioning, and (ii) to discuss the recent advances on schema versioning support in mainstream DBMSs. In particular, the next section gives some basic definitions related to the considered subject. In "Recent Research in Schema Versioning", we present an update on recent research proposals on schema versioning. "DBMS Support for Schema Versioning" surveys the support of schema versioning in the state of the art of the latest database technology. Finally, future work directions and conclusion are provided.

BACKGROUND

The schema versioning technique allows changes to the database schema with continued support of previous schemata and their corresponding data, which are retained without any change. The newly created schema version is (usually) used to accommodate new data insertions, modifications and deletions. This technique neither leads to loss of information nor to obsolescence of existing applications, as they can still work with old schema versions. Further issues related to schema versioning support have been discussed in (Brahmia et al., 2015).

In this section, we illustrate the functioning of schema versioning with a simple example. Let us assume that we have a relational database that contains only an AUTHOR relation with the attributes ID (primary key), NAME, PHONE, and COUNTRY. The first state of this database is shown in Figure 1.

The catalogues store information on the schema S1 of the AUTHOR relation. The table AUTHOR contains two tuples for two authors. Then consider the following schema changes:

```
ALTER TABLE AUTHOR
DROP COLUMN PHONE;
ALTER TABLE AUTHOR
ADD COLUMN EMAIL CHAR(30);
```

In this case, the information related to unchanged attributes (i.e., ID, NAME and COUNTRY) are automatically recovered after executing schema changes. The effects of such schema changes are shown in Figure 2.

In particular, we obtain two schema versions, S1 and S2. The new email information could then be introduced through the following SQL statements:

```
UPDATE AUTHOR
SET EMAIL='aicha@author.tn'
WHERE ID = 1;
UPDATE AUTHOR
SET EMAIL='cristiana@author.it'
WHERE ID = 2;
```

Notice that here, without indicating the schema version under which emails should be introduced, the system is able to automatically determine that these new values have to be inserted in the table corresponding to the second version S2 of the AUTHOR relation.

Moreover, since schema versioning supports all schema versions and their underlying data, it (potentially) allows accessing these data, both retrospectively and prospectively, through user